KATHOLIEKE
UNIVERSITEIT
LEUVEN

DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

RESEARCH REPORT 9939

WORKFLOW MODELLING IN OBJECT-ORIENTED ENVIRONMENTS. REQUIREMENTS, STATE-OF-THE-ART AND FUTURE CHALLENGES

by

S. POELMANS

D/1999/2376/39

Workflow Modelling in Object-Oriented Environments

Requirements, State-of-the-art and Future Challenges

Stephan Poelmans
Katholieke Universiteit Leuven
Department of Applied Economic Sciences
Naamsestraat 69, 3000 Leuven, Belgium
Stephan.Poelmans@econ.kuleuven.ac.be
Phone: +32-16-326881
Fax : +32-16-326732

Abstract

In the past decade, many research efforts have gone into the object-oriented development of information systems and the design of workflow systems. Both domains however, have largely evolved independently. Nevertheless, particular advantages of object-orientedness such as reusability, scalability and portability can be useful in workflow systems. Obviously, strategic advantages that have been gained by analysing business environments with a process view are also important when considering object-oriented developments. In this paper we will describe the necessary requirements to model a business process using the object-oriented approach. Next, we will discuss two approaches to combine workflow systems with object-oriented development: the pure and the mixed approach. Both methods have their strengths and weaknesses and none can claim to be the best solution. We will conclude by giving a short overview of existing models and applications in both approaches.

0. Introduction

Workflow systems and object-oriented (OO) technology have undoubtedly been some of the most important currents of information technology over the past decade. Both domains however, have largely evolved independently, and not much research can be found in which OO principles and concepts have been applied to workflow systems or vice versa. Nevertheless, particular advantages of object-orientedness such as reusability, scalability and portability can be useful in workflow systems. Evidently, strategic advantages that have been gained by analysing business environments with a process view are also important when considering object-oriented applications and developments.

In the following sections we will describe the lack of a process view in present OO analysis and design. The activity-based view of workflow modelling will be shortly discussed and workflow requirements in OO environments will be derived. Next, state-of-the-art efforts to combine both worlds will be presented and discussed. Finally, future research directions will be given.

1. Workflow and OO: some definitions

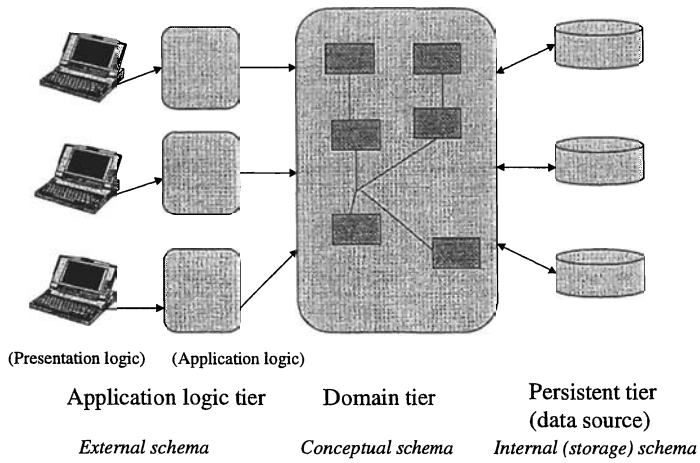
Workflow (Management) Systems have been developed to design, execute, control and adapt business processes in an organisational context. Workflow systems originate from office automation developments in the 80's. Whereas typical office automation applications (like text editors, spreadsheets and databases) are designed to support individual end-users, workflow systems are created to support a process view on business activities. Not only individual users with individual activities but also an entire process with several interdependent users and activities directed towards one common business goal should be supported. In this way, costs can be diminished and/or (customer) services can be improved so that workflow systems can be used as a strategic advantage.

A workflow system can be defined as information technology that can be used to model, enact and adapt business processes (Ellis & Nutt, 93). A business process consists of a number of activities that have to be executed in some order by several end-users in

order to fulfil the business goal. In the modelling phase, the activities, their order of execution and the agents that are responsible for the activities are determined.

One of the principal issues of workflow systems is their ability to cope with changing requirements inside and outside an organisation. In order to make workflow applications more adaptable and easy to maintain, the principles of the object-oriented paradigm can be considered.

Object-oriented system development begins with the identification of the principal object classes that form the building blocks of the final application (Bouzeghoub e.a. 97). Its attributes and associations define the static aspect of an object class. The dynamic aspect represents its behaviour and is modelled by the operations that can be performed on an object. In the object-oriented community, a generally accepted architectural structure is not yet agreed upon. In general, a three-tier architecture is widely approved (Fowler 97). Roughly, the three tiers can be defined as the persistent tier (with data sources that might be stored in flat files or database systems), the domain or control tier (with business semantics) and the application logic tier (with specific application and presentation logic). Alternatively, the three tiers are also called the internal (storage) schema, the conceptual schema and the external schema. In figure 1, the layered architecture is illustrated.



Source: adaptation of Fowler 97, p. 247

Figure 1: different tiers in OO development

The specific contents and meaning of the tiers can vary considerably however. Jacobson e.a. (92) for instance, distinguishes three tiers: the domain tier (that is persistent), the control tier (with business rules and application logic) and the GUI tier (with only presentation logic). Fowler (97) proposes the data tier, the domain tier (with business logic) and the application logic tier (with specific application and presentation logic). In this article, the domain (or control) tier is focussed. The domain tier consists of reusable class libraries that contain business semantics. A class library is a set of reusable classes that address specific programming issues. Classes within a class library are tied together by static relationships.

The OO paradigm models the structure and behaviour of problem domain objects that may be reused and combined for building rapidly business applications. Besides the reusability principle, the OO paradigm also improves the compatibility and openness of an application. Objects can be wrapped and industry standards (like CORBA, DSOM) have been defined to increase cross-platform usability of business objects. In this way, standards stimulate the distribution of information sources in a networked environment.

Although much research is being conducted in the fields of workflow and OO separately, not many efforts have been done yet to apply and use the advantages of OO development and programming in the workflow domain. In the following, we will discuss where workflow modelling should be situated in the OO modelling methodology.

2. Process Modelling in OO environments

In this section, we will first give the necessary requirements to model business processes. Next, we will explain the lack of a process view in OO modelling. Finally, two approaches to combine workflow modelling with the OO paradigm will be compared.

2.1. Requirements of Business Process Modelling

In the literature on workflow modelling, several techniques are proposed to define and represent the structure of a business process (petri-nets, flow charts, etc.). In most cases, the method to be followed is imposed by the vendor of the workflow package (Joosten e.a. 95). The philosophy and theoretical approach that are implicitly or explicitly applied in a particular workflow system, often leaves the designer with no choice. The Action Workflow System for example is built according to the principles of the speech act theory and forces the developer to use the specific diagrams and representations of this theory (Medina-Mora, e.a. 92). The Trigger Model of Joosten (94) uses petri-nets, etc..

Nevertheless, some general requirements can be put forward that are necessary to be able to model a process:

1. Processes, activities and operations need to be defined in a hierarchical manner:
   A process is the highest level of the hierarchy. Process design typically requires a top down decomposition of high level processes into sub-processes down to atomic operations. More precisely, a process might be composed of procedures, which are defined as a limited sequence of activities. Each activity (also called a process step) can be further subdivided into subactivities or subtasks. Activities and tasks might have a different meaning in different organisational theories. In the field of workflow systems however, both terms are often used interchangeably and we will do the same in this article.

   Let us consider a short example to illustrate the hierarchical decomposition. In table 1, the process of granting loans to customers in a bank is decomposed. In the first column, the activities are given. In the second column, activities are further subdivided into subactivities. When a client asks for a loan, the request has to be registered and several subactivities have to be done (such as scanning documents, the creation of a file, etc.). Next, the financial situation has to be analysed and a decision must be made. When the request is approved, the customer is made an offer. The acceptance by the customer means that the offer can be executed.

granting loans to customers	
activities (or process steps)	***subactivities***
1. receive a request	**create a file** **scan documents** **...**
2. analyse the request	**check the necessary documents** **analyse the financial situation of the customer** **formulate an advise** **...**
3. decide on the request	**make a decision** **...**
4. make an offer to the client	**inform the customer** **create a written contract** **sent the contract to the customer** **...**
5. execute the offer	**open an account** **...**
6. follow up	**check incoming documents** **scan incoming documents** **...**

Table 1: decomposition of the loan request process

2. The sequence of procedures, activities and subactivities is crucial:
   The main goal of a workflow system is in fact the automation or support of the co-ordination between activities and between activities and agents. Co-ordination can be defined as the management of dependencies (Malone, e.a. 94). In what way does one activity depend on the results of another activity? The modelling of dependencies constitutes the heart of workflow modelling. The existence of dependencies implies a certain order of execution. Some (sub)tasks cannot be performed before previous tasks have been completed; other tasks need to be executed in parallel, etc. In the previous example, there is a clear sequence of activities. A request must first arrive before the financial situation of the client needs to be analysed; the execution of an offer depends on the result of the decision and the approval of the client; etc.

3. Agents are humans are computer applications:
   Not only the dependencies of activities need to be modelled, but also the interaction between activities and actors (from now on called "agents") needs to be planned ahead. An agent is able to perform different activities, and different agents can

perform a specific activity. Agents can be human end-users or computer applications that perform activities. When human agents execute certain tasks, they might be assisted by computer applications to support them. Only when applications are directly coupled to the workflow system, they are considered as agents. When an application is evoked by the workflow system and when the application performs a certain activity without any intervention of the enduser, it is called an autonomous agent. An agent is called semi-autonomous when it is directly coupled to a workflow system, although an intervention of the enduser is still required.

Let us take the loan request process to illustrate this. Suppose that an enduser has to create a contract for the loan (in activity 4, after a positive decision has been made). If the enduser has to open a text editor, type the text and send it to the customer, then the text editor is not coupled to the workflow system and it is not an agent. On the other hand, if the endusers has to open the text editor and type the text, and if the text editor sends the resulting text automatically to the workflow application, then the text editor is a semi-autonomous agent. It needs to be opened by the enduser but it is directly linked to the workflow application. An autonomous agent could be an application that calculates financial ratios, as soon as an analyst receives a request (in the 2nd activity).

4. Agents are assigned to roles:
   Agents are assigned to activities via the construction of roles. A role defines the responsibility for the performance of a (collection of) task(s). In the example of table 1, the role of *account manager* could be assigned to activity 1, 4, 5 and 6 (opening a file, executing the decision and following up the execution), whereas the role of *analyst* could be assigned to activity 2 and 3 (analysing and deciding).

5. The division of labour in the organisation determines the subdivision in activities and subactivities. In general, the responsibility for one activity should not be assigned to more than one role. One role may be responsible for several activities, but one activity should belong to one role (Kappel e.a. 95).

6. The specification of the business process (or workflow) needs to be a persistent artefact:
   The workflow process is specified as a model in a formal textual and/or visual language. This model specification or definition is used whenever a new workflow instance needs to be created. Each time a workflow instance is created, the persistent workflow model is needed for controlling, supervising and recording the performed activities. Moreover, in order to monitor and improve performance, it is often also required to save the states of instantiated processes that have been enacted. Historical data regarding the actual course of processes can be useful and even necessary to improve the persistent process model.

The dependencies between activities and between activities and agents can be considered as the control logic of business processes. The functional part contains the necessary data and the applications that (partly) perform the activities (the non-human agents). The isolation of the control structure from the data and functional structure is a typical characteristic of workflow systems (Vaishnavi, e.a 97). The process logic is modelled but the functional part is not taken into consideration. In this way, alterations in the progress of the work can be represented in the workflow system by simply adapting the parameters of the process logic. Since the functional part is not considered, it is not clear however how the functional part will be affected by a change in the control logic. A simple change in the process might have considerable consequences for the functional part (Schreyjak 97).

In sum, when modelling a business process (or workflow), an activity-view is advocated. Activities and roles are defined (either or not on a detailed level of granularity) and coupled in a global process. Hence, a business process is divided into a function-oriented part - the activities - and a process-oriented part - the relationship between activities. In a workflow system, the function-oriented part is supposed to be given, whereas the process-oriented part (the process logic) is modelled and supported (Schreyjak 98).

2.2. Business Processes and OO

Workflow systems have been designed to capture and optimise business processes. However, state-of-the-art workflow systems are mostly not object-oriented and they

have intrinsic disadvantages that can be solved by applying the object-oriented methodology.

A first disadvantage has to do with the separation of the process-oriented part from the function-oriented part. Computer agents that support certain activities are regarded as a monolithic black box. This means that adaptations in the process logic might have considerable consequences for the supporting applications. As a result, adaptability is not guaranteed. Because of its modularity and encapsulation, the OO approach is well suited to solve this problem. Required changes in the functional part can be limited to the object classes involved, without jeopardising the consistency of the entire system.

Schreyjak (98) points to another disadvantage of workflow systems. Workflow systems are often used in a heterogeneous infrastructure. Therefore, they need to be platform independent. However, this places a burden on the supporting applications that need to support the activities. When applications are black boxes, it is not sure how well they are suited to be ported to heterogeneous environments. Therefore, Schreyjak advocates the use of components[1] to build autonomous agents. Components can easily be bought and via standards like CORBA, their portability can be guaranteed.

Applying the OO approach can also be considered when modelling the process-oriented part. An OO workflow model can claim the same advantages of OO information systems in general. Using the object-oriented method results in workflow object classes (representing the process logic) that can be reused, ported (for example by making them compliant with international standards) and adapted.

In the following, we will first describe the lack of a process view in current OO development. Next, we will present two approaches to combine the object-oriented method with typical workflow requirements.

2.2.1. The lack of a process view in OO development

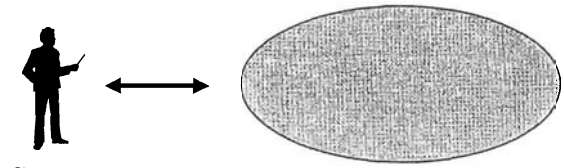
Widespread object-oriented methodologies or specification languages like UML (Booch e.a. 99) or OOA&D (Coad and Yourdon 1991a, 1991b) focus on the construction of a

[1] Components can be object-based or object-oriented. In 2.2.4. components will be defined and discussed more extensively.

structural model, consisting of classes, objects, methods and their relationships. Although use cases and scenarios are used to model the behaviour of the system and its end-users, the behavioural models that finally result will be built on classes and objects of the structural model. Business logic is specified in the form of business rules, which are embedded in objects and classes in the form of methods or operations.

The dynamic behaviour and interaction between object (classes) is modelled in OO development systems by means of several representations like event charts, state transition diagrams, sequence diagrams, etc. Whereas a state transition diagram only models the behaviour of one class, other diagrams (like the sequence or collaborative diagram) model the interaction of several classes. In this way, process logic is implicitly represented and a process-oriented view is in fact not omitted. However, two process modelling requirements that we proposed in the previous paragraph are not met: agents are not assigned to activities (requirement 3 and 4); and process logic is not designed to be implemented as an (persistent) application (requirement 6).

First, via use cases, actors are considered in dynamic representations of the system. However, users described in a use case do not correspond with end-users in a workflow model. In the workflow model, end-users are those who have to interact with the workflow system. In the use case, the user is an actor who represents the environment that interacts with the business process (Jacobson e.a. 95). The example in figure 2 can make this distinction clear. If we consider the example of the loan process of section 2.1., the use case might look as follows:

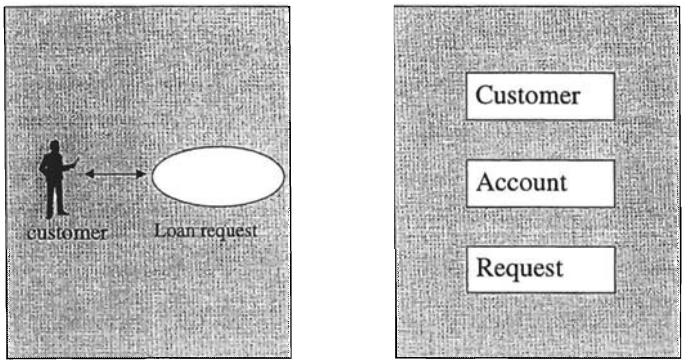


Description:
A loan request starts when the customer contacts
the bank to get a loan. If the customer is an individual,
s/he will go to a local branch office.
If the customer represents a company, s/he will
contact the relation manager of the bank
who is assigned to the company.

Figure 2: the "loan request" use case

The user in the "loan request" use case is a customer, external to the bank, who requests a loan. In the workflow model, this user will not be assigned to a role. Only internal end-users (bank employees) will be assigned to roles to process the loan request. For instance, if the customer belongs to a company, s/he will contact the relation manager of the bank who will then start a new workflow instance.

In the example above, the symbols of the UML method are used: a person, a double-headed arrow and an ellipse. Next to the symbols, a textual description is given. The reason why use cases can also be represented in informal ways (like textual descriptions) is that they are meant to gather requirements and to set the boundaries of the system (Cockburn 97). Use cases describe *what* the system is supposed to do from the actor's perspective. They do not describe *how* the system should be designed and implemented. In order to build a clear architecture of the model, a structure of object (classes) is necessary. In fact, the use cases serve as input for the construction of formalised and structured object classes (Jacobson e.a. 97). The object classes should provide the functionality to support the use cases. In figure 3 a small example is given. The use case "loan request" can lead to the creation of several classes: customer, account and request.

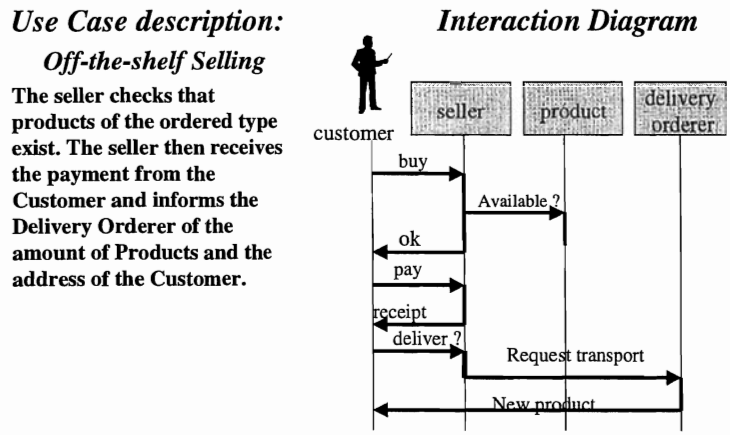


1 use case ... several object classes might result in...

Figure 3: for a use case to be executed, a set of objects is required

Next to use cases, other dynamic representations (like state transition diagrams, sequence diagrams, etc.) are created in the development phase. These diagrams are built on the basis of one or more object classes and they represent processes or procedures. In fact, they are persistent artefacts (that can be stored in a CASE-tool). The process logic however is not explicitly and separately implemented and it should be considered as documentation in the development phase of the way objects and classes can interact in a particular process. Process logic is embedded in classes and changes in business rules need to be done on object(class) level. In this way, an adaptation of object classes is necessary to change the process logic.

In figure 4, an example is given of a use case and the corresponding interaction diagram in UML. The diagram depicts the communication between objects when a customer in a shop orders a certain product.



Source: Jacobson, e.a., 1995, p. 193.

Figure 4: communicating objects in an interaction diagram

The columns in the interaction diagram represent the objects. The messages that the objects send and receive are placed along the vertical axis. The vertical axis indicates a chronological order, going from top to bottom. So, the customer first wants to buy a product and the seller checks if the product is available. If this is the case, the customer has to pay. As soon as the customer gets a receipt of his payment, he asks for the delivery of the product. The diagram ends with the delivery of the product. The interaction diagram clearly contains process logic. However, this logic is only represented in this diagram, in the development phase. Whereas a workflow model is defined and implemented in an application that can be activated at run-time, the interaction diagram is only a visual clarification of how objects can interact at run-time. The process logic of the interaction diagram is not implemented in an application.

In the following paragraph, major challenges that need to be bridged in order to combine object-oriented principles with workflow systems are discussed. First, we will discuss the requirements to design an object-oriented workflow system from scratch (*the pure OO approach*). Next, we will discuss the possibility of combining a traditional workflow system with an object-oriented (or component-based) system that executes the functional part (the activities) of a process (*the mixed approach*).

2.2.2. The Pure Approach : OO workflow modelling from scratch

When studying business processes, process logic and business functionality is at the heart of attention. Both aspects can clearly be considered as business logic. Business logic consists of particular rules (constraints) that only apply in a certain domain. Business rules might only be applicable in one specific application, one process, one enterprise or in a particular domain. Object (classes) that include domain-specific business rules are called business objects (Object Management Group 97). The term "business object" stems from the object-oriented paradigm. Therefore, business objects are strictly spoken object-oriented, which means that they should satisfy OO principles like encapsulation and information hiding. Features like inheritance and polymorphism can also be used. However, in the present relevant literature, there is a conceptual confusion and the term business object is sometimes used to denote any semantic business "building block" or functional unit, whether it is truly object-oriented or not (Fowler 97, D'Souza 99, Bouzeghoub 97, etc.). In this article, a business object is supposed to be object-oriented. In the paragraphs below, we will explain the need of a generic workflow model that is constituted of object classes and that fulfils OO requirements. Such a workflow model consists of a class library with (business) rules to model the decomposition of a business process into activities, subactivities, etc. Such a class library is particularly designed to model the process logic of business processes. The class library can be used to model any process, but it only considers the process logic, i.e. the management of dependencies between activities and between activities and agents.

In the state-of-the-art of the OO literature not many OO workflow models can be found that exist as a persistent class library. In OO development environments, a business process is usually seen as a logical sequence of activities that request services from business objects. When looking at figure 4 for example, the interaction diagram represents the logical sequence of messages that request services from the objects (customer, seller, product and delivery orderer).

Schmidt (98) points at a number of differences in current OO environments between business processes and its constituting elements (which he calls business entities[1]):

1) a business process requests services from business entities, but business entities cannot request a service from a business process;
2) a business entity has a permanent state, whereas a business process vanishes once it has been finished;
3) transaction mechanisms need to be developed in case concurrent changes of business entities' states lead to inconsistency, whereas business processes may be executed in parallel without any provisions.

As mentioned in 2.1., a business process is a hierarchical unit that can be decomposed in a sequence of procedures, activities and subactivities. Translated to object-oriented concepts, a business process can be decomposed in a sequence of services, offered by business entities (and executed in the form of methods and operations). In order to model this hierarchical composition and define a persistent process structure that enables several process enactments, an extra layer should be created. This workflow layer will then consist of class libraries that represent the process logic of business processes. When we consider the three-layered architecture of figure 1, the workflow layer should be positioned between the domain tier and the application logic tier.

Since the application logic tier is only responsible for specific applications, the generic workflow model should be separated from this tier. The workflow model is a class library that can be used to produce several workflow applications. However, since the workflow model provides a process view, it requests services from several business objects that are stored in the domain tier. In other words, the workflow model is the missing link between the business objects and the final workflow application. Business objects are linked, what results in a process view with a higher level of abstraction. Therefore, the generic workflow model should be placed between the application and the domain layer.

[1] A business entity refers to every entity that is below the process level. Business entities are mostly business objects. However, also components or design patterns can be considered as business entities. The difference between business objects, components and patterns will be discussed in section 2.2.4.

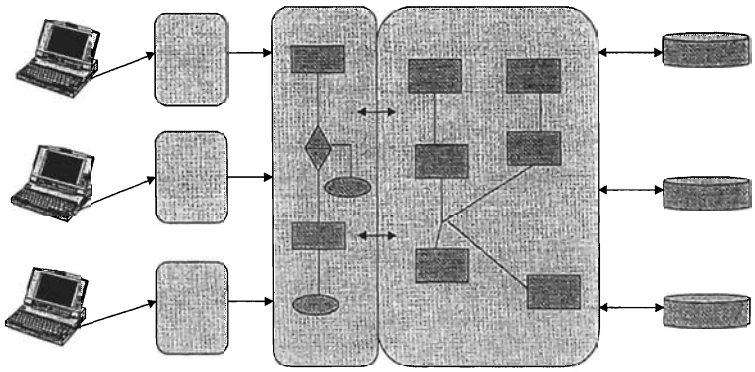


Figure 5: the addition of a workflow layer to an OO layered architecture

In figure 5, the workflow layer is clearly separated from the application logic tier, but it is closely related to the domain tier. As we mentioned in the beginning of this section, the generic workflow model is in fact a class library with business rules, i.e. process logic. Therefore, it fulfils the definition of a business object, which means that it needs to be placed in the domain tier. However, since a workflow model can be developed and maintained independently from the business objects that it combines, it is preferable to place it in a separate tier. Of course, the workflow layer can posses several generic models (realised by several class libraries) with particular process logic.

A generic workflow model can for instance contain an object class "workflow parent" with subclasses that include "activities", "roles", etc. An activity-based workflow application can then be built using the generic model in the workflow layer. It is the responsibility of this layer to map business objects to activities. The assignment of responsibilities to activities is another important requirement that should be foreseen in the workflow layer. In 2.1. we already explained the necessity to link roles to activities. In 3.1. we will present some generic OO models that have been implemented.

Adding a workflow layer with workflow objects means that the advantages of workflow systems (such as a better management of the process and the flexible adaptation of the

process logic) can be achieved, while the link between the functional and the process part is also managed. Both the functional and the process part are encapsulated in separate business objects that communicate by sending messages and triggering methods. As a result, changes in the process logic can easily be communicated to the underlying business objects and vice versa. In the present workflow development tools, the process logic and the functional part are completely isolated (in fact, the functional part is not taken into account), so that for example changes in the process logic might lead to severe adaptations of the functional part.

So far we have considered the *pure approach.* In section 3.1., we will give examples of a generic workflow model that has been implemented. First however, we will discuss a different and more popular approach to combine the functional part with a workflow model: the *mixed approach.*

2.2.3. The Mixed Approach: combining business entities with "traditional" Workflow Systems

In the previous part, we have explained the usefulness of an extra layer to support object-oriented workflow modelling. In the extra layer the process structure should be represented explicitly, in the form of a generic workflow model.

Recently, considerable attention has been paid to a mixed approach that combines existing workflow tools with business entities (business objects, components, etc.). In this approach, a business process is modelled in a "traditional" workflow system (not OO). The result of the modelling effort is a process decomposed according to the organisational division of labour. A process is broken down into procedures, activities, etc. Each lower level is a refinement of the level above.

Central in this scheme is an activity. Human agents or (semi-)autonomous applications can perform activities. As long as the end-user is solely responsible for the activity, the modelling is straightforward. A role is assigned to an activity in the workflow system and the end-user, responsible for the role, can use supporting applications (like spreadsheets, text editors, etc.) to fulfil the objective of the activity. When computer agents (applications) are assigned to a role and directly coupled to the workflow

applications, or when the end-user is partly supported by an application that is automatically evoked, an object-oriented approach can be fruitful.

In most workflow systems, the applications (the business functions) realising the activities are supposed to be given. The application is evoked via a command line interface. The workflow system supplies or transfers parameters and waits for the completion of the application. It offers no support for the modification of the underlying applications that are more or less considered as black boxes (Schreyjak 98). Therefore, adaptability ends at the border of the activities.

The OO paradigm can offer substantial advantages by enabling users to adapt the execution of the (partially) automated activities. Instead of developing applications from scratch, pre-programmed software building blocks (business objects, components or frameworks)[1] can be used to assemble an application without the required knowledge of a traditional programming language. Users can buy and re-use objects with particular business logic, without knowing the exact inner dynamics of the objects. These building blocks can be installed and mapped to a certain activity. In this way, advantages of the workflow paradigm (capturing and improving the process logic) are combined with the advantages of OO.

The coupling of business entities to activities is an important issue. The mapping of business entities to the activities and subactivities of a "traditional" workflow has however not yet gained considerable attention in the scarcely available literature. Both the activities (of the process view) and the business entities (of the OO view) are on a lower level of abstraction than the business process. As mentioned above, business entities are build on the basis of an object model, whereas a business process (a workflow) is created on the basis of the organisational division of activities.[1] So, the process logic and the functional part (the business entities) are developed separately in two different environments. Therefore, the coupling of business entities to activities is not a straightforward matter.

[1] In the next section, we will explain the difference between business objects, components and frameworks. In order to avoid confusion, we simply refer to these concepts as business entities that are on a lower level of abstraction than a business process.

In fact, the business entities contain the data and the applications that are coupled directly to the workflow system to support (sub)activities. If no enduser is required, those applications that execute certain (sub)activities are called autonomous agents. If an enduser is intervening, they are semi-autonomous agents. The business entities that support human agents without being coupled directly to the workflow system, are not discussed here. A small example will make this distinction clear.

If we consider the loan request process of table 1 (section 2.1.), we have the 6 activities in the left column and the corresponding subactivities in the right column. If we look more closely at the first activity, "receive a request", the subactivities "create a file" and "scan documents" can be done by human agents without the intervention of an autonomous agent. The customer visits the bank, requests a loan, and the bank employee creates a request file in the workflow system (he starts a new process instance). Next, the employee scans the necessary documents given by the customer. In order to scan the documents, the bank employee will probably give a command to the scanning application. The application is evoked by the enduser, but if the application exports the electronic documents to the workflow systems automatically, it is called a semi-autonomous agent. If the enduser had to copy the scanned documents and paste them into the workflow application for example, then the scanning application would not a part of the functional part that is scrutinised here.

In the same example, the second activity might require other autonomous agents. Suppose that the employee sends the request via workflow to the analyst of the bank. As soon as the workflow application of the analyst receives the request, the workflow system might evoke an autonomous agent that automatically calculates certain financial indicators of the customer. When the analyst opens the request, he will find some financial ratios that reveal the client's solvability. Here, the application realising the calculation of the financial ratios is evoked without human intervention, so it can be called an autonomous agent.

---

[1] In the "pure approach" (2.2.2.) we suggested to model the workflow model as a business object, in the form of a class hierarchy. However, this class hierarchy is specifically built to represent the process logic and the classes are based on the organisational division of labour.

Whether are not the workflow designer should design activities to fit the non human-agents (the business entities) or whether business entities should be adapted to the process definition remains an open question. If many subactivities can be efficiently performed by computer applications, the workflow activities might be designed to "fit" computer agents. In this case, it can be said that activities are mapped to business entities. However if only a few
(semi-)autonomous agents are required, it might be more appropriate to focus on the division of labour between human agents. In this case, the workflow activities will be designed first, and afterwards business entities will be bought, designed or modified to support certain activities.

For example, in the case of the loan request process, human agents perform many subactivities. So, it is useful to focus on the division of activities among human agents instead of designing activities to match existing business entities. Activities are distributed among endusers who will probably be supported by (semi-) autonomous agents. If only the subactivities "analyse financial situation" and "scanning" are non-human agents that communicate directly with the workflow application, it is obviously more appropriate to adapt business entities to support certain activities.

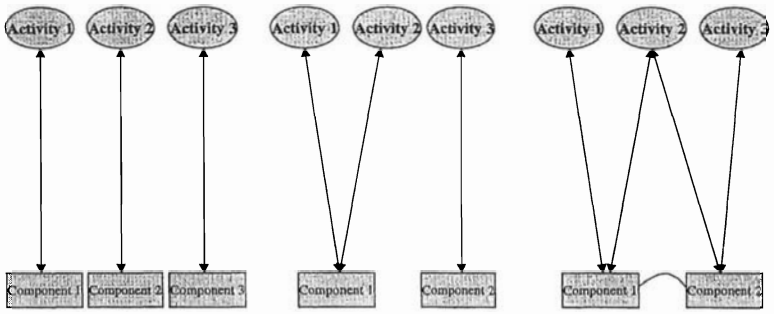
In general, mapping activities to business entities or business entities to activities is in fact a management issue. The choice boils down to the question whether it is feasible to automate (sub)activities. Should a workflow system be introduced to automate the activities as much as possible, or should the workflow system only try to co-ordinate the tasks of the endusers? This obviously is an important discussion that is however beyond the scope of this paper.

When activities are mapped to business entities, activities can be developed so that they fit the business entities. Of course, the management has to consider the consequences for the human agents. Is the intervention of human agents still necessary? What is their remaining task if computer agents execute (sub)activities? A different kind of questions arises if business entities are mapped to activities. In this case, the workflow process is designed, and business entities are then adapted to support certain activities.

The business entities (objects) that are used by the business process can be mapped in several ways to the activities. One business entity can be mapped to exactly one object, one business object can be mapped to support several activities or several business entities might be required to support one activity. Mapping one business entity to exactly one activity however improves adaptability for the IT responsible or even the enduser. In this approach, system developers can plug in new versions of available business objects or components to adapt the functional part of the process. Moreover, an adaptation of an activity will only lead to the adaptation of one business entity.

Nevertheless, Schreyjak (97) points to difficulties in scalability when an entity is mapped to exactly one activity. This approach might enhance adaptability, but it leads to a large number of small business entities (components or business objects), which places a heavy burden on the workflow engine. On the other hand, if several business entities need to communicate to execute a particular activity, the IT responsible needs to take into account the dependencies between the collaborating entities. In this way, business entities cannot simply be plugged in, and adapting the functional part can demand considerable time and/or knowledge. In sum, a trade-off between adaptability and scalability exists.

In figure 6, the trade-off problem is illustrated when business entities are mapped to activities. In case 1, components can be plugged directly to activities, which increases adaptability. New versions of components can easily be installed and changes in an activity only require the adaptation of the corresponding component. However, components will likely be fine-grained so that scalability might be troublesome. In case 3, an activity might need several components. Therefore, components need not be fine-grained, which can improve scalability. However, the dependencies between components require extra attention. Users cannot easily plug in existing customised components, which diminishes adaptability. In case 2, one component can be used for several activities, but each activity can only evoke one component. In this way, scalability is not a problem and components need not be combined to support activities. However, a change in one activity requires a change in the supporting component, which might have consequences for the other activities that rely on that same component. In sum, all possibilities have their advantages and disadvantages and no general best solution exists.



Case 1: 1 to 1 mapping

Case 3 : many to many mapping

Case 2 : one to many mapping

Figure 6: Mapping components to activities

Despite the mapping problem, the mixed approach gains considerable attention in the literature. In section 3, we will illustrate two implementations. So far we suggested that a business process should be decomposed into activities. Business entities can be used to support these activities. Above, we already indicated that business entities might be business objects. Next to business objects, a lot of attention has recently also been paid to components, design patterns and frameworks. Although components and certainly design patterns and frameworks exist on a higher level of granularity then business objects, they should not be confused with business processes. In the next section, we will shortly explain these concepts and indicate how they relate to workflow modelling.

2.2.4. Business Objects, Components, Patterns and Frameworks

Recently, many efforts have been put in the development of components, (design) patterns and application frameworks. Particularly design patterns and frameworks model parts of entire (enterprise-wide) applications. In this way, they may be used to implement business processes. Nevertheless, significant conceptual differences exist between design patterns, frameworks and workflow models. In the following paragraphs, these concepts are positioned and distinguished.

Components vs. business objects

A component is a set of elements (mostly business objects) organised into groups (sometimes called categories) which perform functions commonly needed within business applications (Bouzeghoub, e.a. 97). Components are useful to provide a level of granularity that is higher than that of object classes or business objects. Components are the building blocks of applications. As opposed to business objects, components need not be object-oriented. True object-oriented components support features like a class structure, inheritance or polymorphism. If one of these features is missing, components are said to be object-based. The level of granularity of a component is determined to optimise the reusability of components (Vandenbulcke 98).

Following the ideas of Schmidt (98), components can be considered as business entities but not yet as business processes. As in the case of business objects, the mapping problem remains unsolved. Providing a component for each activity is conceptually easy and might stimulate adaptability, but it burdens the workflow engine at run-time.

When discussing components in the context of workflows, we consider them as (small) business applications that are ready to be used and coupled to one or more activities. At the analysis level, components are mostly called packages. A package is then "a container for any piece of development work you want to treat as a unit" (D'Souza, e.a. 99). Since the focus in this article is on modelling workflow applications, we will not consider how components have been developed.

Design Patterns and frameworks

Design Patterns systematically name, motivate, and explain a general solution that addresses a recurring problem (Mowbray 97). Patterns are expert common-sense solutions for a particular problem. They are reusable for a range of similar problems in a variety of different contexts. A framework is a specification of a generic solution to a large-grained problem. Frameworks can use design patterns and a particular design pattern can be used in several frameworks (Vandenbulcke 98). A framework can in fact be compared with a class library. Just like in a class library, classes are also tied together by static relationships in a framework. However, a framework also contains some application logic that couples the various classes. Class libraries on the contrary, do not contain any specific application (business) logic.

Design patterns and application frameworks are conceptually related to business processes in the sense that they contain business logic that is on a higher hierarchical level than business entities. In other words, patterns and particularly frameworks consider the combination of several business entities. Patterns and frameworks are pragmatic solutions to recurrent problems that are close to actual business applications. However, they do not necessarily define applications that are enterprise-wide. Both single-user applications and multi-user applications (including business processes) are envisioned. The scope of patterns and frameworks ranges from the application-level to the enterprise-level (Mowbray e.a. 97).

Patterns that define a business process can be called workflow patterns. A workflow pattern is the solution to one particular process. As a result, it should not be confused with a generic workflow model. The same argument accounts for frameworks. A framework can have such a scope that it covers an entire business process. It remains however a generic solution to a specific problem with several variation points that can be adapted to fulfil the particular needs of a context (Matheussen 98).

In this article, we suppose a hierarchical decomposition according to the level of granularity. Business objects are seen as a class structure with business logic. Components are considered to be on a higher level of granularity. They are functional units that can combine several business objects. Components need not be strictly object-oriented. Next, design patterns are built on the combination of several components or business objects. Several design patterns can finally be integrated in one framework. In the literature however, there is a kind of conceptual confusion. Not all authors will follow the hierarchy that is proposed in this paper. For example, some use the term "component" to indicate any logical software unit. To them, a framework should also be considered as a component, etc. Nevertheless, the core idea is that there is a hierarchical decomposition going from a low level of granularity (a small functional business unit) to an entire application. The construction of an application via components, patterns and frameworks is called "component-based development" (Matheussen 98, e.a.).

3. Existing OO workflow models: state-of-the-art

So far, we have delimited the specific requirements for workflow modelling when using an object-oriented approach. Next, we discussed two approaches: modelling a general

OO workflow model or combining existing (non-OO) workflow systems with business entities.
Finally, we discussed related concepts that do not posses the idiosyncrasies of a workflow system. In the following we will illustrate some approaches that have been implemented. Examples of pure OO workflow modelling and of the mixed approach will be given.

The list is by no means exhaustive. It is supposed to illustrate the ideas that have been presented in this article. It gives an overview of the models that are frequently commented on and referred to in the literature.

3.1. OO workflow models

We will first shortly mention several approaches that have been proposed. Finally, we will illustrate $TriGS_{flow}$ more in detail because it defines a single workflow class hierarchy that corresponds well with the ideas discussed in section 2.2.2.

3.1.1. InConcert, TOWE and CORBAflow

In the InConcert system, a workflow definition of a particular business process belongs to a class (InConcert 97). Activities, roles, etc. are modelled via subclasses. A subclass inherits the attributes of its super class and can specialise by adding more. In fact, no generic workflow type or class structure is foreseen. InConcert is not an OO development tool and it doesn't posses a layered architecture. It is particularly developed to model business processes. In the layered model, we would situate the InConcert processes in the domain and application layer. It offers the possibility to build a complete application with specific process (or business) logic and a GUI (the presentation logic).

As opposed to InConcert, the TOWE system defines a set of classes that provide basic mechanisms for workflow execution (Papazoglou, e.a. 97). The base classes should be seen as a generic workflow structure. When a specific workflow model (a workflow type) is designed, it has to inherit from the base classes. The workflow model is finally implemented as a class. As with the InConcert system, all elements of the workflow model are implemented in the body of the class. It is clear that the base classes can be placed in the extra workflow layer discussed in 2.2.2.

In the CORBAflow system, Crawley (e.a. 97) describes an object model based on the OMG architecture. Relationships between elements of a particular workflow type are defined explicitly. A workflow type is not a single object class but a set of (CORBA) objects that are related. Modelling a complete workflow by subclassing a complete type is therefore not possible. The idea of a generic workflow class or model is not applied.

3.1.2. TriGS$_{flow}$

TriGS$_{flow}$ is an approach to implement workflow applications on the basis of the object-oriented and rule-based paradigm. In fact, it is based on the object-oriented database system GemStone.

Kappel e.a. (95) have elaborated an object-oriented generic model to implement workflow processes. The model should be situated in the extra workflow layer that we discussed in 2.2.2.

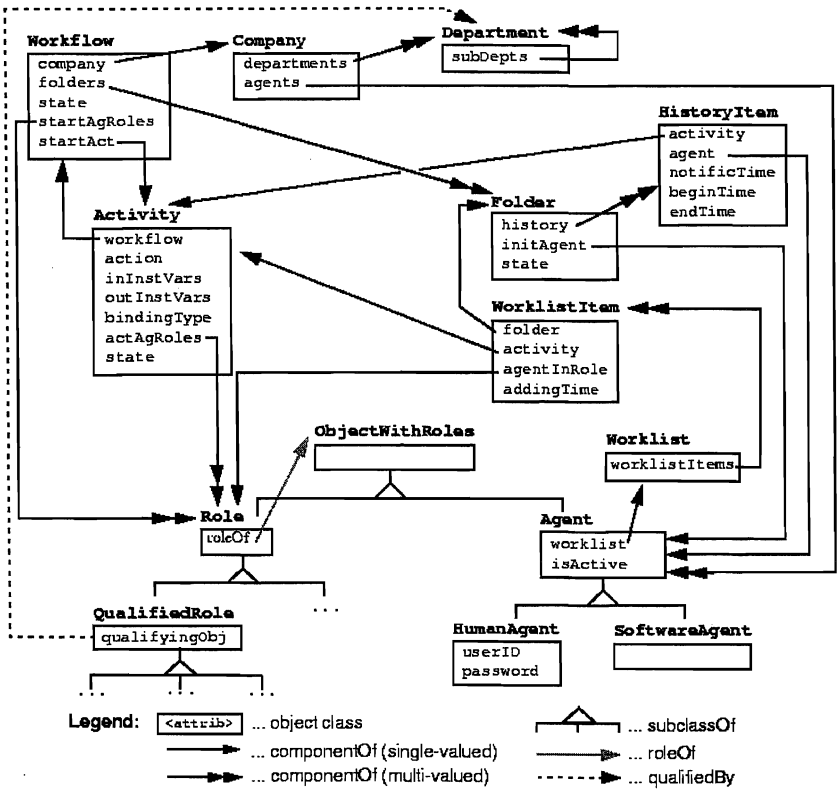


Figure 7: The Class Library of TriGS$_{flow}$

$TriGS_{flow}$ is realised as a class library consisting of 18 base classes. The dynamic behaviour of $TriGS_{flow}$ is based on Event/Condition/Action (ECA) rules. These rules are employed for activity ordering and agent selection. The rules can be attached to specific classes or defined independently of any class hierarchy. Rules can be activated at different levels of granularity ranging from the object instance to the object class level. The *event* of an activity-ordering rule constitutes the end of one or more preceding activities. The *condition* checks whether the succeeding activity has to be performed or not. The *action* of a rule notifies the agent responsible for the succeeding activity.

Besides the execution order of activities, data need to be exchanged between agents to realise co-operation. Folders contain the necessary data for performing an activity. They are created at the start of a new workflow instance (cf. the subclass Folder). If an agent is asked to perform an activity (via the action part of an ECA rule), a new worklist item is inserted into his/her/its worklist. The worklist item defines the activity to be performed and includes a reference to the relevant folder.

$TriGS_{flow}$ allows integrating both internal applications implemented within the same object-oriented environment as well as external applications such as a text processor. The subclass *Softwareagent*, uses a starting command to evoke external applications. Calling methods associated to classes evokes internal object-oriented applications.

$TriGS_{flow}$ clearly realises the idea of an extra workflow layer. Agents are defined as either human or autonomous and they are assigned to roles. Applications are mapped to activities via start commands or methods. Since business entities need not be mapped to a traditional workflow system, the mapping problem of figure 5 is not applicable.[1]

3.2. Mixed approaches

Two systems are presented: the Surro workflow system and Schreyjak's approach.

3.2.1. Surro

Leymann (95) proposes an approach to build an object-oriented workflow by removing all flow dependencies between business objects. For each activity, a business object

exists and a dedicated control object manages the flow control (the process logic). Leymann uses a traditional workflow system as the control object. He calls it the "heavy-weight" approach.

Leymanns' ideas have been realised in the Surro workflow system. This workflow engine uses an Object Request Broker to call business objects methods to perform an activity (Schreyjak 97). For each activity a corresponding component exists and the engine is used as a central control unit.

3.2.2. Schreyjak

Schreyjak points to the scalability problems when mapping one activity to one business object (as is the case in Surro). He proposes to couple a component-oriented system with a workflow system in such a way that several components need to collaborate to support one activity (many to many mapping). The workflow system takes over the process-oriented part of the business process and co-ordinates the execution of the sequential order of activities. The component system manages the flow between components to execute a specific activity.

The approach is based on a compound document architecture. The workflow system supplies the component system with the necessary compound documents for the execution of an activity. The compound documents are composed of business objects. The activity is called an application in the component system. The use of process contexts enables the coupling of the applications to the activities in the workflow system. A process context is a set of activities in a process associated to each other by a certain purpose. The workflow system provides the component system with many contexts that contain certain conditions. Therefore applications are "process aware" and can adapt to a specific process. An example of a context is the set of all activities that belong to a particular department in a company. In the context, next to other options, particular default values for data-entry are defined. The component system receives the process context and can deliver context (or process) sensitive applications.

---

[1] As long as object classes are mapped to activities via methods, objects communicate via message passing and the mapping problem is not an issue. Also, when external applications need to be mapped to activities, the external applications are considered to be given and the mapping problem doesn't apply.

Schreyjak's approach aims at designing a workflow application that can be easily modified by its users in two respects. First, by using (traditional) workflow systems, the adaptation of the process logic should be improved. In non-OO workflow systems, no attention is paid to the underlying applications of the activities. Secondly, users can modify applications by buying or creating business components that can be coupled to the activities in the workflow system by using process contexts.

4. Conclusions and Future Research

Present workflow systems lack adaptability in several ways. One of the most obvious limitations is that workflow applications can only be installed on a limited number of platforms. Besides that, control is limited to the process logic. The applications that are necessary to perform the activities are evoked via commands and they are considered as black boxes. In order to overcome these disadvantages, the OO paradigm promises to be fruitful.

Based on a literature study, we have tried to explore the possibilities to combine the OO paradigm with the workflow paradigm. We investigated two major approaches. First, generic workflow models could be developed from scratch, in a pure object-oriented way. In this approach, the generic model is based on a class structure so that it possesses typical OO characteristics such as inheritance, polymorphism, etc. In the second approach, business processes are still modelled in existing (non-OO) workflow tools, but activities are coupled to business entities like business objects, components, patterns or frameworks. Both approaches have their specific problems and advantages.

In the pure OO approach, OO advantages such as reusability, scalability and compatibility can be accounted for. When a generic OO workflow model is created, the integration with existing OO business applications is straightforward. When using traditional workflow systems however, the coupling between business entities (mostly components) with activities is an unsolved issue. The mapping problem implies a trade-off between adaptability and scalability. Of course, when using traditional workflow systems, a generic workflow model need not be defined and the developer can make use of an extensive body of existing knowledge. It seems that in practice, most research efforts are being put in the mixed approach.

Future research need not focus on one of the two approaches. Both approaches could be combined when generic object-oriented modelling is implemented in current workflow tools. In this way, workflow tools could deliver applications that are more easily ported to several platforms. This would increase the reusability of generic and particular workflow models and the coupling with business entities could be solved by OO techniques such as message passing and event handling.

Another research direction regards the extension of design patterns and frameworks. Since business processes are quite similar within particular business domains, research in this area can lead to a knowledge base of usable workflow patterns and frameworks. Workflow patterns and frameworks can more easily be developed and applied if the workflow model or tool is object-oriented or at least "object-enabled".

Bibliography

Booch, G., J. Rumbauch & I. Jacobson, The Unified Model Language User Guide, Addison-Wesley, 1999, 482 pp

Bouzeghoub, M., Gardarin G. & P. Valduriez, Object Technology, Concepts and Methods, 1997, 382 pp.

Cockburn, A., Structuring Use Cases with Goals, in *Journal of Object-Oriented Programming*, Sept-Oct 1997.

Coad P. & E. Yourdon Object-Oriented Analysis, 1991.

Coad P. & E. Yourdon Object-Oriented Design, 1991.

Crawley, C. & O. Bukhres, Failure Handling in CORBAFlow: A CORBA-based Transactional Workflow Architecture. In *Proceedings of the fifth International Conference on Database Systems for Advanced Applications*, Melbourne, Australia, April 1997.

D'Souza, D.F. & A. C. Wills, Objects, Components and Frameworks with UML, The Catalysis Approach, Addison-Wesley, 1999, 785 pp..

Ellis C.A. & G.J. Nutt, Modelling and Enactment of Workflow Systems, Technical report, Department of Computer Science, University of Colorado, 1993.

Fowler, M., Analysis Patterns, Reusable Object Models, Addison Wesley Longman, 1997, 357 pp.

InConcert Process Designer's Guide. Software Release 3.6., InConcert, inc. 1997.

Jacobson, I., Christerson, M., P. Jonsson & G. Övergaard, Object-Oriented Software Engineering, Addison Wesley, 1992, 528 pp.

Jacobson, I., M. Ericsson & A. Jacobson, The Object Advantage. Business Process Reengineering with Object Technology, 1995, 347 pp.

Joosten, S. Trigger modelling for workflow analysis. In *Proceedings of CON '94: Workflow Management, Challenges, Paradigms and Products*, October 1994, München, pp. 236-247.

Joosten, S. Werkstromen : een overzicht, in *Informatie*, jaargang 37, nr. 9, pp. 519-528.

Kappel, G., P. Lang, S. Rausch-Schott, & W. Retschitzegger, Workflow management based on objects, rules and roles, In *Bulletin of the Technical Committee on Data Engineering*, March 1995,*18(1)*, pp. 11-18.

Leymann, F., Workflows make Objects Really Useful, In *Proceedings of the 6th International workshop on High performance Transaction systems (HPTS)*, 1995.

Malone, T. W. & K. Crowston, The Interdisciplinary Study of Co-ordination, In *ACM Computing Surveys*, Vol. 26, No. 1, March 94, pp.87-119.

Matheussen, D., Business frameworks en ontwikkelen met componenten, In *Informatie*, februari 1998, pp. 14-25.

Medina-Mora, R., Winograd, T., Flores, & Flores, F. The Action workflow approach to workflow management technology. In *Proceedings of the Conference on Computer Supported Co-operative Work '92*, New York, Nov. 1992, pp. 281-288.

Mowbray, T, J. & R. C. Malveau, CORBA Design Patterns, John Wiley & Sons Inc., 334 pp.

Object Management Group, CORBA Component Model RFP, OMG document orbos/97-06-12, July '97.

Papazoglou, M., Delis A. & A. Bouguettaya, Class Library Support for Workflow Environments and Applications, In *IEEE Transactions on Computers*, Vol. 46, No. 6, June 1997.

Schmidt, M. Building Workflow Business Objects, In *OOPSLA'98,* Business Workshop IV, 17 pp.

Schreyjak, S., Coupling of Workflow and Component-Oriented Systems, In *Second International Workshop on Component-Oriented Programming*, 9 pp.

Schreyjak, S., Using Components in Workflow Activities, In *Proceedings of the Second and Third International Workshop on Business Objects*, 1998, 12 pp.

Vaishnavi, V., Joosten, S. & B. Kuechhler, Representing Workflow Management systems with Smart Objects, 1997, 7 pp.

Vandenbulcke J., Met componentensoftware naar de wendbare onderneming, In *Informatie*, februari 1998, pp. 6-12